CIS 455/555: Internet and Web Systems

RPCs

October 27, 2021

Plan for today

- Hadoop and HDFS ✓
  - Architecture
  - Using Hadoop
  - Using HDFS
  - Beyond MapReduce
- Remote Procedure Calls ⬅ NEXT
- Web Services

From Clusters to Fully Distributed Computing

- Let’s formalize the process of communicating among decoupled software components on different machines!

- Initially: client-to-server (or server-to-server)

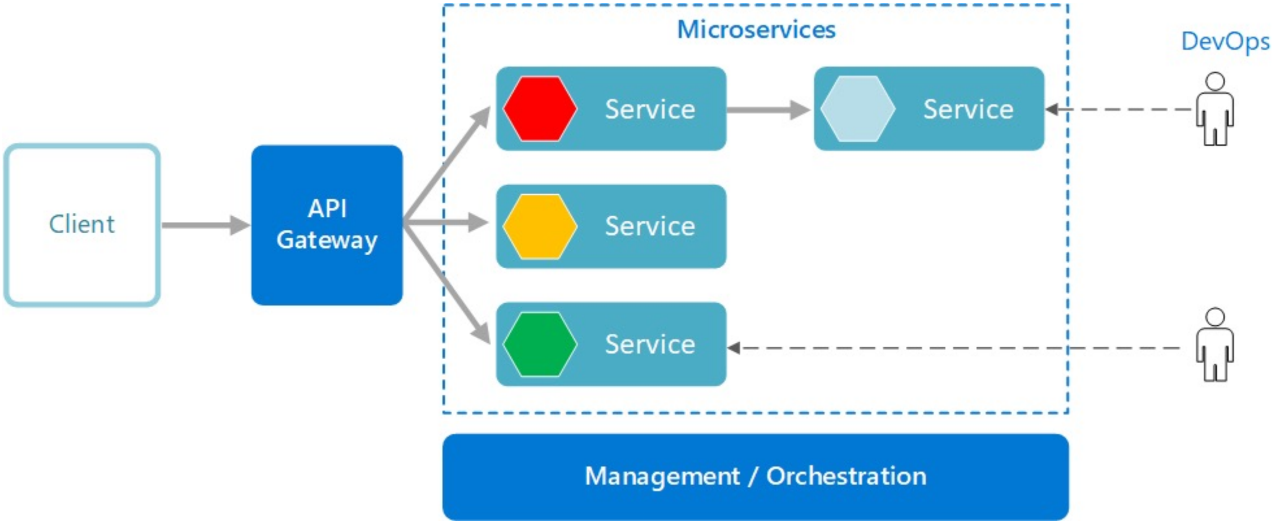
- Next module: distributed messaging in peer-to-peer form

Modern Software and Modularity

- In the cloud era – many apps have distributed components, often written by different teams in different languages

- How do we make this modular, possible to "scale out", and able to tolerate changes?
  - Microservices – an informal term for applications built in components, each invoked via HTTP
  - (A generalization of your HW2M2 channel services)

Building a Client Application



https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices

- A client app is a composition of multiple independent services
  - often called via a library + web services through a gateway
- Services are typically separate code bases
  - deployed independently
  - in own languages, built by own teams
  - loosely coupled, usually persist their data or state independently of others

Nginx – a Popular API Gateway

- Suppose we have an array of different microservices…
  - For modularity, each may have its own separate HTTP server
  - Perhaps in a container, or on a different server in a cluster

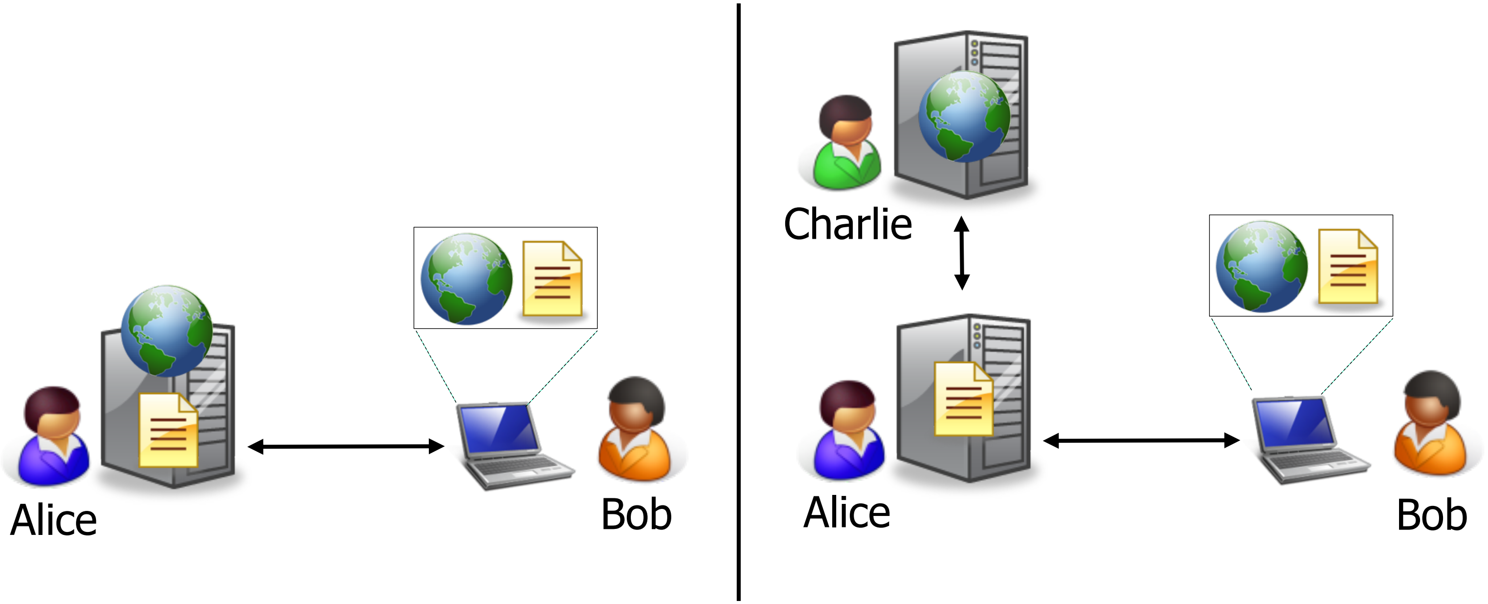
- Ideally, we would map them all to related paths in the same domain
- The solution – nginx: proxy server + HTTP server
  - Map / to Sparkjava
  - Map /webapi to separate Node.js server
  - Map /data to Minio in Docker

Who Uses Microservices?

- Amazon – they use AWS internally
- Netflix – uses nginx
- Dropbox – uses nginx
- Uber
- Ebay
- Comcast cable

- etc.

Key Component: Web Services



- An application or set of APIs that is accessible to other applications over the web
  - Examples: Google Search, Google Maps API, Facebook Graph API, Amazon Web Services, ...
  - Triggered via HTTP
- With Spark Framework: you can define these as route handlers!

Today: Essentials of Web Services

- Foundations: remote procedure calls
- Web service basics
- Real-world REST services

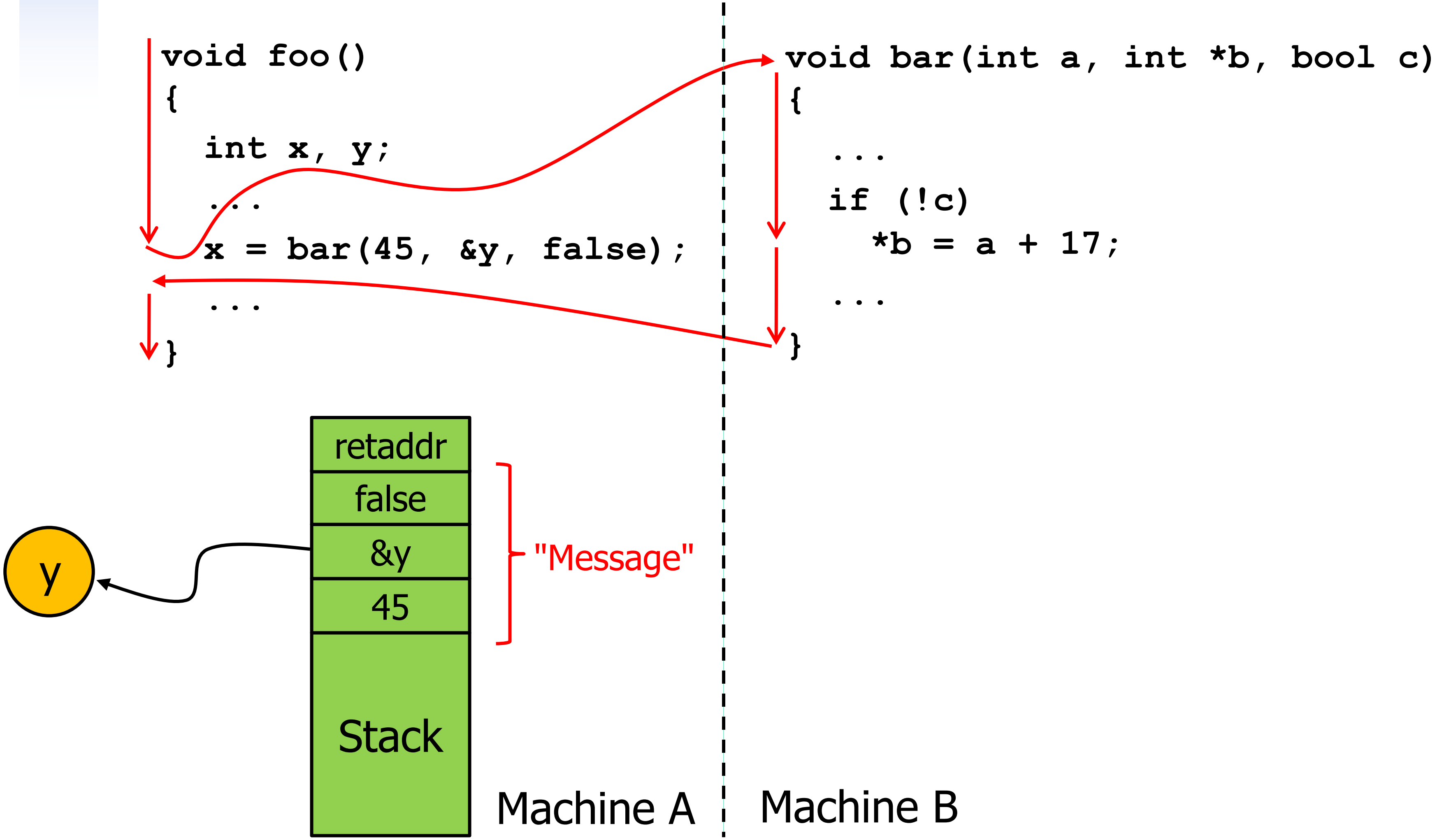
Plan for today

- Hadoop and HDFS ✓
- Remote Procedure Calls ⇦ NEXT
    - Abstraction
    - Mechanism
    - Stub-code generation
- Web services
    - REST vs SOAP
    - Real REST services

Motivation for RPCs

- Coding your own messaging is hard
  - Example: Look up a name on our directory server
  - Assemble the message at the sender, parse at the receiver...
  - Other things too: which service? How to get return values?
- Let’s hide this in the programming language and middleware
  - Similar strategy works great for many other hard or cumbersome tasks, e.g., memory management
  - Wouldn't it be nice if we could simply call a function lookup(name) in the client code, and it executes remotely on the name server?
  - That is the abstraction provided by Remote Procedure Calls

The intuition behind RPCs



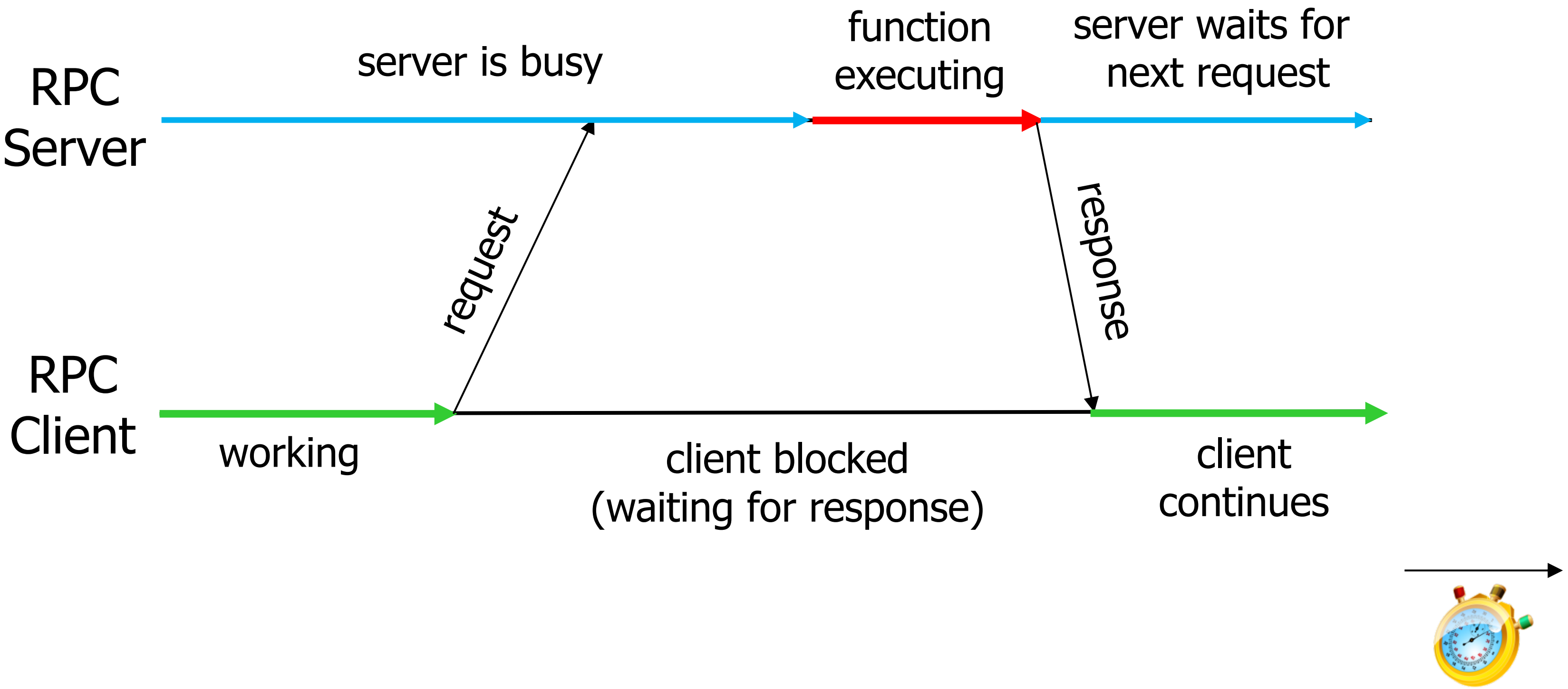
```
void foo()
{
  int x, y;
  ...
  x = bar(45, &y, false);
  ...
}
```

```
void bar(int a, int *b, bool c)
{
  ...
  if (!c)
    *b = a + 17;
  ...
}
```

Remote Procedure Calls

- Remote procedure calls have been around forever
  - Implementation examples: COM+, CORBA, DCE, Java RMI, ...
- An RPC API defines a format for:
  - Initiating a call on a given server, generally in a reliable way
    - At-most-once, at-least-once, exactly-once semantics
  - Sending parameters (marshalling) to the server
  - Receiving a return value - may require marshalling as well
  - Different language bindings may exist
    - Java client can call C++ server, Fortran client can call Pascal server, …
- Traditionally: RPC calls are synchronous
  - Caller blocks until response is received from callee
  - Exception: One-way RPCs
  - Modern RPC frameworks include support for async calls

RPC visualized

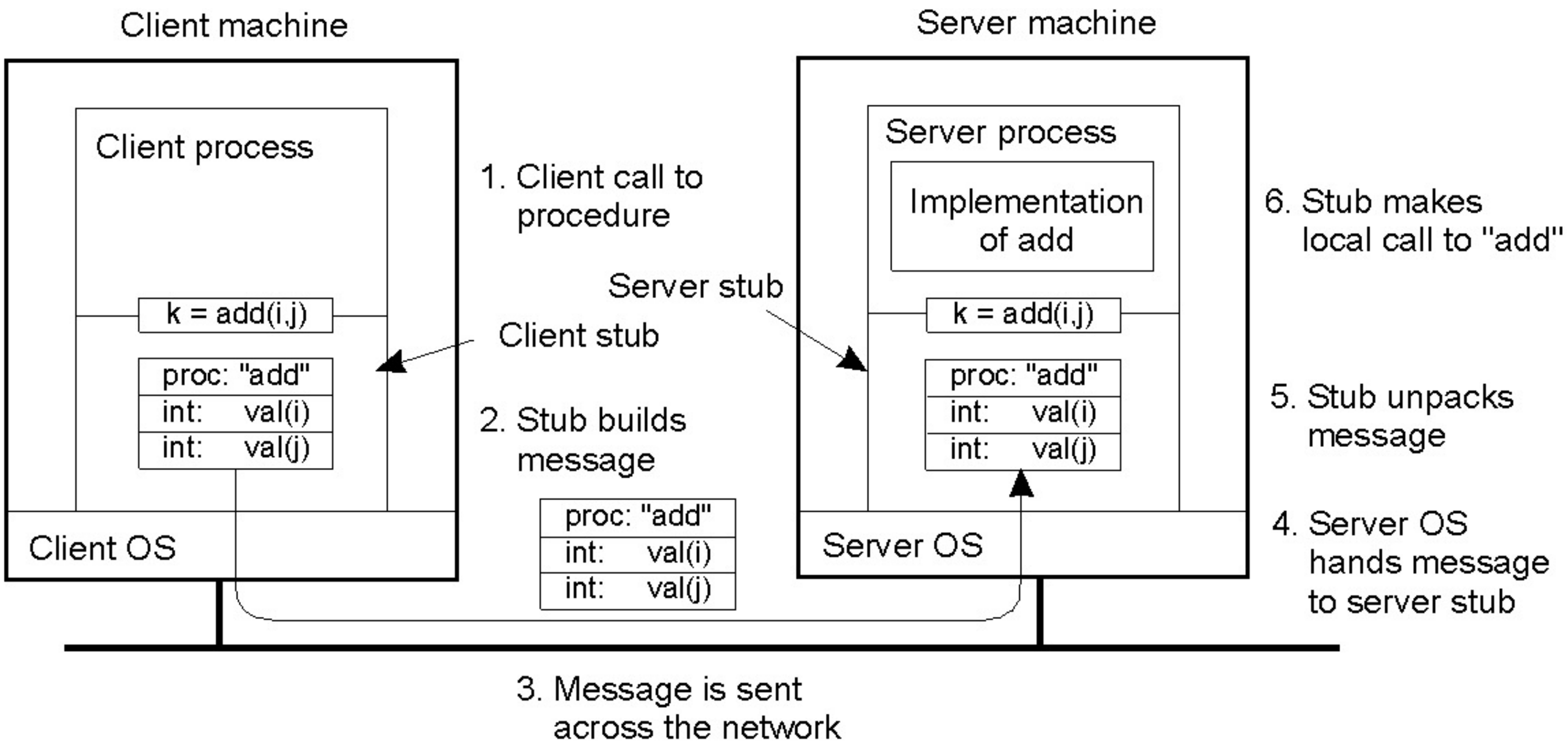


How RPC generally works

- You write an application with a series of functions
- Some of these functions will be distributed remotely
- You call a stub-code generator, which produces
  - A client stub, which emulates each function F:
    - Marshals all the parameters and produces a request message
    - Opens a connection to the server and sends the request
    - Receives response, unmarshals+returns F's return values, status
  - A server stub, which emulates the caller on the server side:
    - Receives a request for F with parameters
    - Unmarshals the parameters, invokes F
    - Takes F's return status (e.g., protection fault), return value, marshals it, produces a response, and sends it back to the client
    - Waits for the next request (or returns to the server loop)

Passing value parameters

- Steps involved in doing remote computation through RPC



RPC components

- Generally, you need to write:
  - Your function, in a compatible language
  - An interface definition, analogous to a C header file, so other people can program for F without having its source
  - Includes annotations for marshalling, e.g., [in] and [out]
  - Special interface definition languages (IDLs) exist for this
- Stub-code generator takes the interface definition and generate the appropriate stubs
  - (In the case of Java, RMIC knows enough about Java to run directly on the source file)
- The server stubs will generally run in some type of daemon process on the server
  - Each function will need a globally unique name or GUID

An example

```
module StockObjects
{
  struct Quote {
    string symbol;
    long at_time;
    double price;
    long volume;
  };

  exception Unknown{};

  interface Stock {
    // Returns the current stock quote.
    Quote get_quote() raises(Unknown);

    // Sets the current stock quote.
    void set_quote(in Quote stock_quote);

    // Provides the stock description, e.g. company name.
    readonly attribute string description;
  };

  interface StockFactory {
    Stock create_stock(in string symbol, in string description );
  };
};
```

http://java.sun.com/developer/onlineTraining/corba/corba.html

What are the hard problems with RPC?

- Resolving different data formats between languages (e.g., Java vs. Fortran arrays)
- Reliability, security
- Finding remote procedures in the first place
- Extensibility/maintainability
- (Some of these might look familiar from when we talked about data exchange!)

Plan for today

- Hadoop and HDFS ✓
- Remote Procedure Calls ✓
  - Abstraction ✓
  - Mechanism ✓
  - Stub-code generation ✓
- Web services ← NEXT
  - REST vs SOAP
  - Real REST services

(W3C) Web Services

> "A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards."
>
> *http://www.w3.org/TR/ws-arch/*

- Key elements:
  - Machine-to-machine interaction
  - Interoperable (with other applications and services)
  - Machine-processable format
- Key technologies:
  - SOAP and REST
  - WSDL (Web Services Description language; XML-based)

Web Services – The Vision

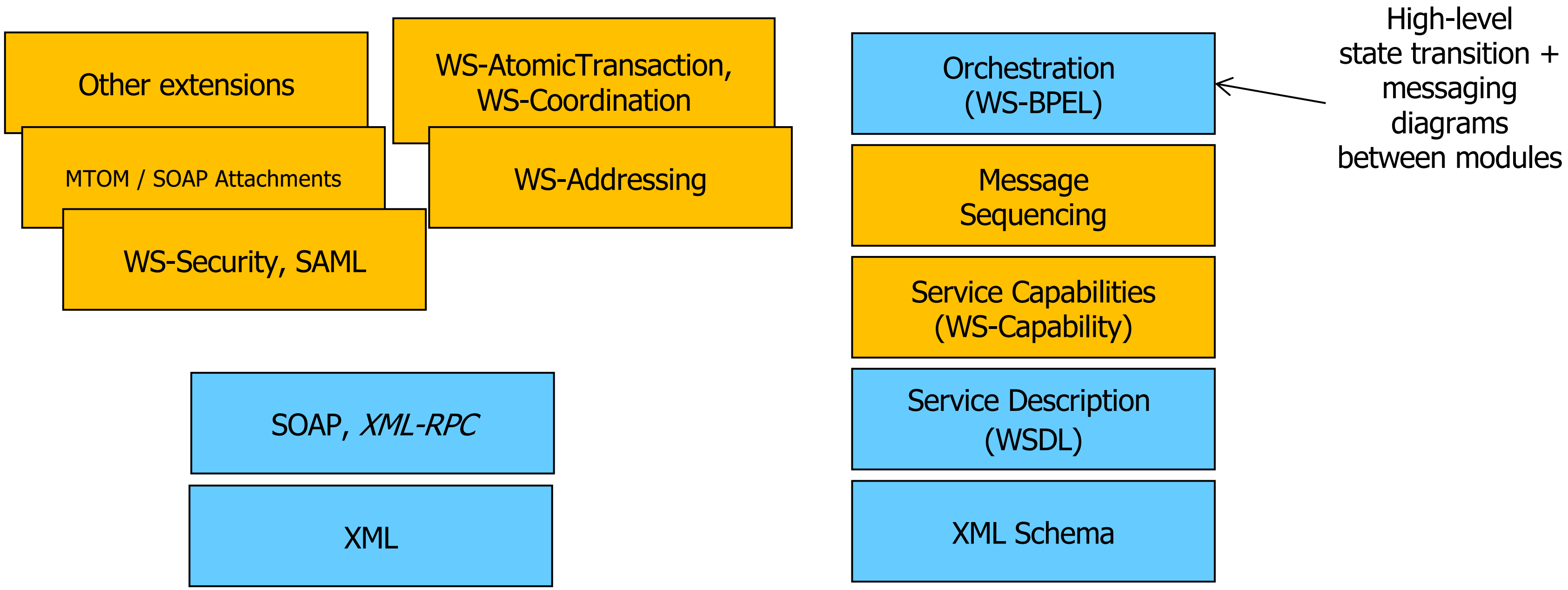
- Goal: Provide an infrastructure for connecting components, building applications similar to hyperlinks between data
  - Directly on the web – it can be a means of assembling content in a *mashup*
  - More commonly today: we build libraries that make RPCs
- A distributed computing platform for the Web
  - Internet-scale, language-independent, upwards-compatible where possible
- This one is based on many familiar concepts
  - Standard protocols: HTTP
  - Standard marshalling formats: XML-based, XML Schemas
  - All new data formats are XML-based

The “Standard” for Web Services

Three parts:

1. “Wire” / messaging protocols
   - Data encodings, RPC calls or document passing, etc.
   - We will discuss: SOAP and REST
2. Describing what goes on the wire
   - Schemas for the data
   - We have already discussed: XML Schema
3. “Service discovery”
   - Means of finding web services

The 'protocol stacks' of web services



Wire Format Stack

Description Stack

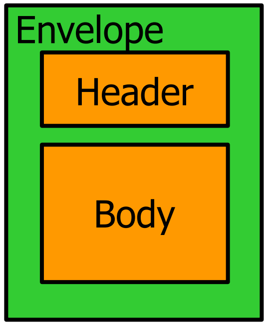
Enhanced + expanded from a figure from IBM's "Web Services Insider", http://www.ibm.com/developerworks/webservices/library/ws-ref2/index.html

REST and SOAP

- Example: Access AWS from your program
  - Example: Launch an EC2 instance, store a value in S3, ...
- Simple Object Access protocol (SOAP)
  - Not as simple as the name suggests
  - Full of features: well-defined encodings, security/authentication, QoS, failure handling, etc.
  - XML-based, extensible, general, standardized, but also somewhat heavyweight and verbose
- Representational State Transfer (REST)
  - Much simpler to develop than SOAP
  - Web-specific; lack of standards

Simple Object Access Protocol (SOAP)

- One example of a messaging protocol
- XML-based format for passing parameters
    - Has a SOAP header and body inside an envelope
    - Has a defined HTTP binding (POST with content-type of application/soap+xml)
    - A companion SOAP Attachments Protocol encapsulates other (MIME) data
    - The header defines information about processing: encoding, signatures, etc.
        - It's extensible, and there's a special attribute called mustUnderstand that is attached to elements that must be supported by the callee
    - The body defines the actual application-defined data



Envelope

Header

Body

Making a SOAP Call

- To execute a call to service PlaceOrder:

```
POST /PlaceOrder HTTP/1.1
Host: my.server.com
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: nnn

<SOAP-ENV:Envelope>
   …
</SOAP-ENV:Envelope>
```

SOAP Return Values

- If successful, the SOAP response will generally be another SOAP message with the return data values, much like the request
- If failure, the contents of the SOAP envelop will generally be a Fault message, along the lines of:

```
<SOAP-ENV:Fault>
  <SOAP-ENV:Code>
    <SOAP-ENV:Value>SOAP-ENV:Sender</SOAP-ENV:Value>
  </SOAP-ENV:Code>
<SOAP-ENV:Reason>
  <SOAP-ENV:Text xml:lang="en">Sender Timeout</SOAP-ENV:Text>
</SOAP-ENV:Reason>
</SOAP-ENV:Fault>
```

Example: SOAP envelope

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
xmlns:SOAP-ENC='http://schemas.xmlsoap.org/soap/encoding/'
xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
xmlns:xsd='http://www.w3.org/2001/XMLSchema'>
<SOAP-ENV:Body>
<PutAttributesRequest xmlns='http://sdb.amazonaws.com/doc/
2009-04-15'>
<Attribute><Name>a1</Name><Value>2</Value></Attribute>
<Attribute><Name>a2</Name><Value>4</Value></Attribute>
<DomainName>domain1</DomainName>
<ItemName>eID001</ItemName>
<Version>2009-04-15</Version>
</PutAttributesRequest>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Sample request

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<PutAttributesResponse>
<ResponseMetadata>
<RequestId>4c68e051-fe45-43b2-992a-
a24017ffe7ab</RequestId>
<BoxUsage>0.0000219907</BoxUsage>
</ResponseMetadata>
</PutAttributesResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Sample response

Source: http://awsdocs.s3.amazonaws.com/SDB/latest/sdb-dg.pdf

Extensions

- WS-Notification
    - WS-BaseNotification
    - WS-Topics
    - WS-BrokeredNotification
- WS-Addressing
- WS-Transfer
- WS-Eventing
- WS-Enumeration
- WS-MakeConnection
- WS-ReliableMessaging
- WS-Reliability
- WS-RM Policy Assertion
- WS-Security
- WS-Policy
- WS-PolicyAssertions
- WS-PolicyAttachment
- WS-Discovery
- WS-Inspection
- WS-MetadataExchange
- …

Representational State Transfer (REST)

- Another example of a messaging protocol
- Not really a standard – a style of development
    - Data is represented in XML, e.g., with a schema
    - Function call interface uses HTTP Requests
        - GET/POST/PUT/PATCH/DELETE
        - Server is to be stateless
    - And the HTTP request type specifies the operation
        - e.g., GET http://my.com/rest/service1
        - e.g., POST http://my.com/rest/service1 {body} adds the body to the service

Example: REST

Invoked method

Response elements

https://sdb.amazonaws.com/?Action=PutAttributes

Parameters
&DomainName=MyDomain
&ItemName=Item123
&Attribute.1.Name=Color&Attribute.1.Value=Blue
&Attribute.2.Name=Size&Attribute.2.Value=Med
&Attribute.3.Name=Price&Attribute.3.Value=0014.99

Credentials
&AWSAccessKeyId=<*valid_access_key*>
&Version=2009-04-15
&Signature=[valid signature]
&SignatureVersion=2
&SignatureMethod=HmacSHA256

&Timestamp=2014-01-25T15%3A01%3A28-07%3A00

Sample request

<PutAttributesResponse>
<ResponseMetadata>
<StatusCode>Success</StatusCode>
<RequestId>f6820318-9658-4a9d-89f8-b067c90904fc</RequestId>
<BoxUsage>0.0000219907</BoxUsage>
</ResponseMetadata>
</PutAttributesResponse>

Sample response

Source: http://awsdocs.s3.amazonaws.com/SDB/latest/sdb-dg.pdf

Other Parts of the W3C Stack

- WSDL – Web Services Description Language – also allows us to define functions, modules, etc.
- Orchestration languages allow us to script sequences of executions into "business processes"

Key Take-Aways

- REST maps function operation types to HTTP methods (GET, PUT, etc.)
  - And creates a conceptual space of paths and subpaths
- SOAP is part of a broader, strongly typed set of interfaces
  - Defines input + return types, exceptions, and more
  - "Envelope" allows for extra info, exceptions, and more
- Easy to automatically generate calls in W3C SOAP spec from any language, given that it is a full specification including types
- REST has no formal published spec: probably requires libraries from the service provider

Plan for today

- Web services ✓
  - REST vs SOAP ✓
  - Real REST services ← NEXT
- Information retrieval
  - Basics
  - Precision and recall
  - Taxonomy of IR models